Dynamic Algorithm Selection in Parallel GAMESS on a Chip Multithreaded Processor

Lakshminarasimhan Seshagiri[1], Masha Sosonkina[1], and Zhao Zhang[2]

[1] Department of Electrical and Computer Engineering
Ames Laboratory
Iowa State University
Ames, IA 50011 USA
sln,masha@scl.ameslab.gov,

[2] Department of Electrical and Computer Engineering
Iowa State University
Ames, IA 50011 USA
zzhang@iastate.edu

Abstract. Multi-core and multi-threaded processing environments have become the norm in the generic computing environment and are being considered added an extra dimension to the execution of any application. Inclusion of this dimension in any application adaptation scheme has shown to provide a great deal of improvement in the application performance. General Atomic and Molecular Electronic Structure (GAMESS) used for ab-initio molecular quantum chemistry calculations has been chosen to test such an adaptation scheme using the middleware NICAN. GAMESS has two different implementations of Self Consistency Fields(SCF) methods which uses either the disk IO or the memory and switching between these has shown to provide great performance gains. In this paper, we test and show the he suitability of the adaptation between the two SCF implementations on a Chip Multithreaded processor.

1 Introduction

General Atomic and Molecular Electronic Structure(GAMESS) performs ab-initio molecular quantum chemistry calculations [2] to perform a wide range of Hartee-Fock (HF) wave function (RHF,ROHF and UHF) calculations . Using Self Consistent Field (SCF) method, GAMESS iteratively approximates solution to the shrodinger equation that describes the basic structure of atoms and molecules. GAMESS uses two different SCF implementation methods, direct and conventional. The direct algorithm recomputes integrals on the fly for each iteration mainly consuming physical memory and CPU resources. Conversely, the conventional diskbased algorithm calculates integrals once, stores them on disk, and reuses during iterations resulting in a heavy disk I/O usage.

Numerous GAMESS calculations have parallel implementations utilizing distributed resources like memory and disk storage. The scalability of GAMESS is aided by the use of a native communication layer DDI [7] that takes advantage of shared memory on symmetric multiprocessors (SMP) and reduces the remote data access bottleneck. [13] has shown that a parallel job with computing resources distributed over different nodes is faster than a job running on a single node. In [9] [4], it was shown that this could be due to the inability of the communication layer implementations to handle the shared memory access. Hence, a single processor per node scattering was used for better utilization of the networking hardware. Also, a conventional GAMESS job run in parallel is slower than the same job run sequentially due to IO bottlenecks caused by simultaneous access of the physical files. It was also shown in [12] that while running concurrent scattered GAMESS jobs, if only a single conventional job is run, we are able to achieve better performance. This modification to the GAMESS job run can be done such that the execution algorithm can be selected at the start of every job run as well as during the execution of the job. GAMESS source code modification is not a feasible option here and hence a generic middleware like NICAN was used to perform the above mentioned adaptation [1]. The integrated middleware tool monitors system resources, analyzes job performance and invokes adaptation handlers to improve resource utilization and application performance.These tests were carried out in a SMP cluster environment.

A multi-core and multithreaded processor can be used as a single execution environment in itself instead of a SMP cluster. The execution semantics change in such an environment. This paper looks at the suitability of the adaptation strategy on a Chip Multithreaded processor and is organized as follows. Section **??** gives an introduction to the multi-core and multi-threaded processors and their characteristics. Section 3 explains the dynamic switching in GAMESS using the NICAN middleware. Section **??** talks about the switching algorithm and its usage in a multi-threaded environment. Section shows the results obtained by using the algorithm selection strategy on a Chip multithreaded Processor.

2 Introduction to a Multithreaded environment

Advances in processor technology have enabled us to utilize not only multi-core processors but also processor that are capable of running multiple threads concurrently on a single core. Most of the clusters used nowadays are multi-core machines. We need to analyze the characteristics of such processors first in order to exploit them for usage in the switching algorithm. One of the main characteristics is that of the memory access. It is of paramount importance in a computationally intensive application like GAMESS, as it determines the performance of the user job. Most of the clusters that we use can be viewed as a form of NUMA architecture. NUMA (Non Uniform memory access) is the design used where each processor in a multi processor environment is provided with a separate memory space and data is being shared between different memory banks This needs to be handled using separate hardware and software since the data can be distributed over different processor memories and coherency between this data needs to be maintained. The communication cost will also play a significant role in this design. Each node in a cluster can be considered equivalent to a processor in NUMA architecture, but with a coupling that is not as tight as that in NUMA. The nodal latency and bandwidth in clusters is much worse than a normal NUMA machine. Hence when we run an IO intensive application like the conventional GAMESS job, it is very likely to bog down the channel thereby resulting in slower execution times .

Consider a multi-threaded processor like the Sun T2 Niagara, where the communication and data sharing between the different cores has to be viewed in a different way. The T2 Niagara processor is a CMP (Chip Multi-Threaded Processors) that has 8 cores and is capable of running 8 threads on each core simultaneously. Each one of these threads acts as a virtual processor to the outside world. Thus the user application sees itself running on a 64-processor machine with access to each and every one of them. Each of these virtual processors includes all the architecturally required components to execute a thread that includes registers both general purpose and special), integer and floating point execution units and can handle interrupts. Thus each processor contains a separate instance of the user state. The proximity of the shared resources like cache to the virtual processors executing the code has a tremendous impact on the execution times. The communication cost between processors executing the code on different CMT processors is not similar to the cost when the code is executed on the same CMT. However, by executing the code on the same CMT, we give rise to resource contention issues. This is very similar to the NUMA model and gives us a direction to code our applications. This model would define GAMESS execution on such CMTs and the endeavor of this paper is also to look at different mapping strategies for GAMESS on a T2 processor.

3 Dynamic Switching in GAMESS using NICAN middleware

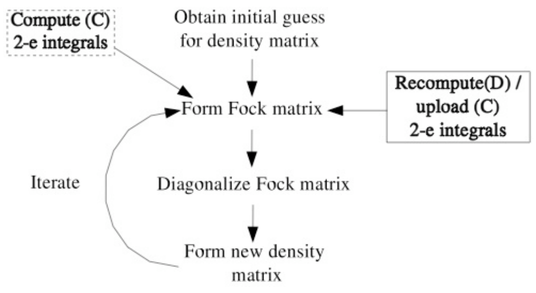

Fig. 1. Illustration of the SCF algorithm as *conventional* (C) and *direct* (D) implementations.

The SCF algorithm is one of the most computationally intensive parts in the GAMESS execution. Selection of the correct electronic structure calculation routine has a very big effect on the overall calculation and calculation time. The SCF algorithm is shown in Figure 1 in a diagrammatic form. The main difference between the two implementations (conventional and direct) is the handling of the two-electron (2-e) integrals. In the direct SCF method, the 2-e integrals are recalculated for each iteration. This method is more computationally intensive but it avoids any I/O bottleneck. In the conventional SCF method, the 2-e integrals are calculated once at the beginning of the SCF process (box with dashed border in Figure 1) and stored in a file on disk for subsequent iterations. Since the SCF algorithm (Figure 1) is of iterative nature, switching between conventional and direct implementations may be possible in an arbitrary SCF iteration. The switching between the direct and conventional methods is carried out using the NICAN middleware. A middleware is specifically used here in order to decouple the application from having to make any adaptation decisions during the application execution while at the same time limit the application involvement only to the invocation of the adaptation handlers.

The GAMESS adaptation scheme using NICAN has been explained in detail in [1]. Every GAMESS job that starts, first checks the peer jobs and in case there is one other conventional job running, it changes the job execution to the direct method. This adaptation is done using the control port(part of NICAN middleware) during the preprocessing stage of the job. Once the execution starts, the control port receives information from the GAMESS code through the daemon and determines whether the adaptation should be done or not. This control port is generalized to different molecules and computational systems. The control port combines system and application related information that it gathers both at startup (static information) and during the actual run of the application (dynamic information). The system information includes information like the number of processors on which the job has to be run, the maximum time that the job requires to run, the memory that is available with the system and can be utilized by the job etc. The application related information includes the estimated ideal time for running one iteration of the job, the actual iteration time, the maximum number of iterations that need to be run, the upper bound of time required for each iteration and the average iteration time over the iterations that have been completed. The formal algorithm has been described below.

```
$t_N$ = actual time taken for iteration N
$t_N^u$ = upper bound for the time per iteration N
$m$ = Average iteration time over N iterations
$\Delta t_0 = t_0^e \ t_0$
**if** ($t_i > t^u$ OR $t_i > m + \Delta t_0$) **then**
  **if** (SCF is conventional) **then**
    switch to direct
  **else if** ((no peer conventional jobs) AND (enough memory)) **then**
    switch to conventional
  **end if**
**end if**
```

The experimental results obtained for this algorithm on a SMP have been given in [1]. We use the processor affinity property to test this algorithm on multi-threaded and multi-core environment where each core is capable of running multiple execution threads. Processor affinity is the process of allocating each process to a specific core such that all the resources available to that core is utilized by that process. We can use this affinity model to schedule each individual GAMESS job or its parts to a set of virtual processors. The experiments were conducted on a T2 Niagara CMT, which is capable of running 64 threads simultaneously. These experiments have helped us to understand the execution semantics on such multi-threaded processors.

4 Experiments and Results

We consider the effects of running parallel versions of GAMESS on a single T2 Niagara processor but after ensuring that the jobs are run on different processor sets. We are considering that only GAMESS jobs are run on this platform and multiple GAMESS jobs request for the available resources for execution. The test platform is a box that contains a single T2 Niagara processor with a 16GB main memory that is shared between all the cores. Each of the 64 virtual processors operates at a frequency of 1.167 GHz. The processor has a 16-way set associative L2 cache of 4MB. It is banked eight ways so that each of the physical cores has enough memory bandwidth. We chose the Luciferin and Ergosterol molecules for performing the RHF SCF calculations. For Luciferin, GAMESS converges in 15 iterations with the conventional SCF algorithm requiring storing files of almost 3.5GB or with Direct SCF algorithm consuming 5.65MB of main memory. Similarly, the Ergosterol molecule converges in 15 iterations with the conventional SCF algorithm requiring almost 22GB of disk space for files and []MB of main memory for performing the Direct calculation. These values are estimated by the GAMESS check module. All the GAMESS calculations start off as Conventional by default.

We first benchmark both these molecules by running single jobs on different sets of processor thread combinations. For the Luciferin molecule we can see that as we increase the number of threads the execution time reduces for both the conventional and direct mode of calculation. On the other hand, for the Ergosterol molecule, the conventional mode of calculation starts to take more time as we increase the number of threads. The direct mode is much faster for the Ergosterol molecule. A conventional GAMESS job can be characterized into two parts. One is the part where the integral files are written. This part is very heavy on disk I/O. The second is the RHF SCF calculation using the integral values calculated in step one. For every iteration, the GAMESS job has to fetch the integral values from the files. If the files are small enough to fit in the main memory, it gives us a distinct advantage in terms of performance, as we dont need to calculate the integrals again for each iteration. On the other hand, if the integral files are so large that they cannot be fit completely in the main memory, the amount of page faults increases dramatically as we increase the number of threads. This slows down the application considerably. In such cases, the direct mode of operation is the best possible method of calculation. This is illustrated in the benchmarking results that we have obtained. The main memory for a Niagara processor is 16GB. For the Luciferin molecule, the entire file written in the conventional mode can be accommodated in the main memory and this reduces the latency of the virtual processors in getting the necessary integral data from the files. On the other hand, the Ergosterol files cannot be accommodated in the main memory and hence the conventional mode of operation degrades in performance as we increase the number of threads.

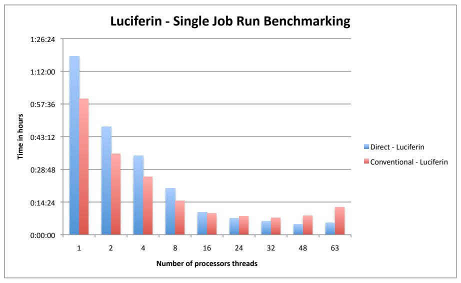

Fig. 2. Luciferin : Single Job Benchmark

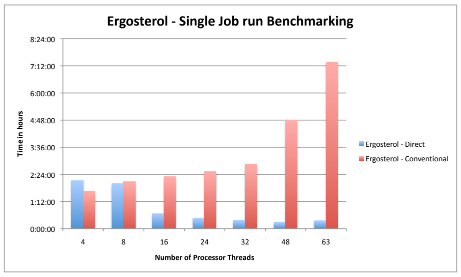

Fig. 3. Ergosterol : Single Job Benchmark

One more thing to note is that the kernel itself is run on one of the 64-processor threads. When we create processor sets, we are not allowed to assign all the 64 threads to different processor sets since a single virtual processor is required for running the kernel. In such a scenario, if we assign 63 threads to execute a single GAMESS job, the job takes more time than when run with 48 threads. This is clearly visible in the benchmarking tests carried out on the Luciferin (Conventional and Direct) and Ergosterol (Direct) execution. The kernel requires more resources than allocated for performing other activities like context switches, network activities of the machine etc. Also, the hardware in a core is shared between the eight virtual processors. Hence having only a single virtual processor free and the others assigned for GAMESS is not good from the performance perspective.

The next set of experiments involved running simultaneous parallel GAMESS jobs by partitioning the virtual processors equally, between the jobs but ensuring that the physical cores associated with these threads are not divided between the jobs. This ensures that the hardware resources associated with the physical cores are used in completeness by the GAMESS job and not by any other process. We ran these tests, first by using two GAMESS jobs (One Luciferin Molecule and one Ergosterol molecule) and three GAMESS jobs (Two Luciferin Molecules and one Ergosterol molecule). We compare the performance between an original GAMESS with a conventional mode and GAMESS in which the NICAN middleware triggers a dynamic algorithm selection of the SCF algorithm. The results for the simultaneous execution of parallel GAMESS jobs shows that we obtain nearly 50 percent gain in performance due to the switching algorithm. The gain is less when we have a 4 processor thread run but increases as we increase the number of processor threads allocated to each job to 31. The adaptation in these tests includes both static and dynamic adaptation. We first introduce the larger molecule Ergosterol in the system and then introduce the smaller molecule Luciferin to observe the performance of the adaptation algorithm. It has been observed that at lower processor thread allocation, the smaller molecule (Luciferin) run is transformed into a Direct method of execution due to the presence of a peer Ergosterol molecule but then switches back to conventional mode dynamically to ensure a faster run time. For larger processor thread sets, the Direct mode of execution is faster than the Conventional mode for Luciferin and it completes its execution in the Direct mode itself. A similar trend is observed for the larger molecule (Ergosterol) as well. As we increase the number of processor thread allocation to Ergosterol, it becomes more and more resource constrained due to the presence of other Luciferin molecules and switches dynamically to use a Direct method of execution.

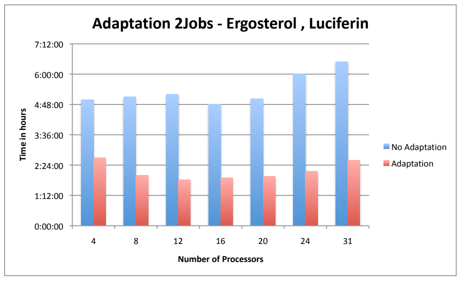

Fig. 4. Two Simultaneous Parallel Jobs Execution

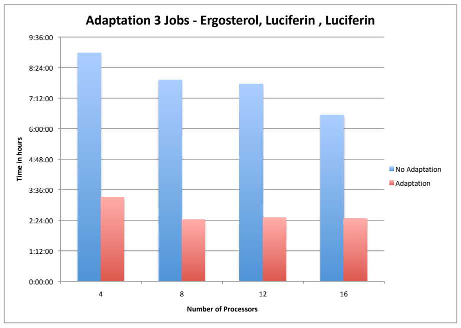

Fig. 5. Three Simultaneous Parallel Jobs Execution

5 Conclusions and Future Work

The main focus of this work was to demonstrate that the adaptation algorithm first introduced in [1] could be used to obtain performance improvement in GAMESS when the execution environment is shifted to a multi-threaded environment. We have shown that as a result of static and dynamic adaptations in GAMESS, the execution can be several magnitudes faster than the non-adaptive GAMESS execution. On a chip multithreaded processor like the Niagara, the I/O becomes a bottleneck as we start increasing the number of threads for a Conventional mode of execution. In such cases, it was observed that the Direct mode is the best way of execution. Also, it has been seen that the Conventional mode will be faster than the Direct mode, when the entire integral file can be placed in the main memory of the processor and the number of threads being used is also small. As we increase the computing power by increasing the number of processor threads, the overhead of multiple threads accessing the files starts to degrade the performance of a Conventional execution method and we start to get better performance in the Direct mode. This difference is essential in getting the adaptation to work on such processors.

One thing to note here is that these experiments are not a means to benchmark the T2 Niagara processor for suitability to run large parallel legacy codes like GAMESS. The endeavor was to test the adaptation capabilities of GAMESS using NICAN middleware on a Chip Multithreaded processor. It would be interesting to see how the application adaptability behaves when we have a cluster of such multithreaded processors. The GAMESS-NICAN adaptation strategy is a very generic adaptation strategy that can be reused with any parallel application. In the future we would like to develop multiple adaptation control strategies for usage on such processors. One of these could be to change the allocation of threads dynamically from a single core to span multiple cores that would allow for higher hardware processing power to the application. Such application independent parameters can be exploited during application runtime in these processors to obtain higher performance gains.

References

1. Nurzhan Ustemirov, Masha Sosonkina, Mark S. Gordon and Michael W. Schmidt. Dynamic Algorithm Selection in Parallel GAMESS Calculations Parallel Processing Workshops, International Conference on, vol. 0, no. 0, pp. 489-496, 2006 International Conference on Parallel Processing Workshops (ICPPW'06), 2006.
2. M.W. Schmidt, K.K. Baldridge, J.A. Boatz, S.T. Elbert, M.S. Gordon, J.H. Jensen, S. Koseki, N. Matsunaga, K.A. Nguyen, S. Su, T.L. Windus, M. Dupuis, J.A. Montgomery. General Atomic and Molecular Electronic Structure System. *Journal of Computational Chemistry*, 14, 1347-1363(1993).
3. B.M. Bode and M.S. Gordon. Macmolplt: a graphical user interface for GAMESS Journal of Molecular Graphics and Modeling, 16, 133-138 (1998).
4. X. Chen, D. Turner. Efficient Message-Passing within SMP Systems. Recent Advances in Parallel Virtual Machine and Message Passing Interface, 10th European PVM/MPI conference, Venice, Italy, pg 286-293 (October 2003).
5. F. Jensen. Introduction to Computational Chemistry. Wiley, Chester UK, 1999
6. W. D. Norcott and D. Capps. Iozone Filesystem Benchmark. `http://www.iozone.org`
7. R. M. Olson, M. W. Schmidt, M. S. Gordon, A. P. Rendell. Enabling the Efficient Use of SMP Clusters: The GAMESS/DDI Model, Proceedings of the 2003 ACM/IEEE conference on Supercomputing, p.41, November 15-21, 2003
8. M. Sosonkina. Adapting Distributed Scientific Applications to Run-time Network Conditions. In J. Dongarra, K. Madsen and Jerzy Wasniewski, editors, Applied Parallel Computing, State of the Art in Scientific Computing, 7th International Workshop, PARA 2004, Revised Selected Papers, volume 3732 of Lecture Notes in Computer Science, pages 745–755. Springer, 2006.

9. M. Sosonkina, S. Storie. Parallel performance of an iterative method in cluster environments: an experimental study. In *Proceedings PMAA 2004, Marseille*, October 2004.
10. S. Storie. Aspects of Communication Subsystem Analysis for Distributed Scientific Applications. Masters Thesis, University of Minnesota Duluth, May 2004.
11. E.H. White, F. Capra, W.D. McElroy. The Structure and Synthesis of Firefly Luciferin *J. Am. Chem. Soc.*, 83(10), 2402-2403(1961).
12. N. Ustemirov, M. Sosonkina, M.S. Gordon, M.W. Schmidt. Concurrent Execution of Electronic Structure Calculations in SMP Environments. In *Proceedings HPC 2005*, April 2005.
13. N. Ustemirov, M. Sosonkina. Efficient Execution of Parallel Electronic Structure Calculations on SMP Clusters. Minnesota Supercomputing Institute Technical Report umsi-2005-227, University of Minnesota, 2005